How to Create a Java SOAP Client

Introduction

This article describes how to create a SOAP[1] Client using Java and the Axis Java libraries. Axis is an Apache project that provides Web Services libraries.

The article includes two samples: a "DOM-style" client that sends and receives DOM[2] objects, and a "marshaling-style" client that uses marshaled objects. The DOM example also illustrates how to write data to an Excel spreadsheet. (DOM and marshalling are described below.)

The article is a simple "how-to" – it explains the steps needed to send and receive SOAP messages. The article does not explain what's going on behind the scenes; for in-depth information please see the "Sources of Information" section at the end of the article.

The reader should have basic understanding of XML and know how to write and run a Java program.

SOAP and Web Services

SOAP is a standard for sending and receiving documents to a Web Service. A Web Service is a server that can read such messages, process their data, and send responses.

A Web Service is analogous to a Web Application. Web Applications receive requests from a Web Browser, such as Internet Explorer, and send back HTML documents in response. In this case, the client application is the Web Browser; it is responsible for formatting the user request, sending it to the server, and receiving and interpreting the response. Web Services clients go through the same cycle of creating a request, sending it to the server, and receiving and processing the response. In this case, the response is simply an XML file, whose contents can be processed as needed by the client.

A Web Services Description Language (WSDL) file fully describes how to use a Web Service. In other words, the WSDL is the public API for a Web Service—it describes the operations and messages available to the client application. There are open source Java utilities that generate client applications based on a WSDL. In other words, one can automatically generate stubbed-out applications that take care of sending SOAP message data provided by the programmer, and returning the server response. (There are also .NET tools that generate client applications.)

Axis

Axis is the Apache Software Foundation's third generation SOAP framework. Using Axis one can create server and client applications. Axis allows the programmer to create two types of client: a client that sends and receives DOM objects, or a client that reads marshaled objects.

DOM Messages

DOM stands for Document Object Model: a Java API for processing XML trees. DOM is published as a set of interfaces by the World Wide Web Consortium (W3C). There are other libraries that implement DOM and allow the programmer to create or inspect XML trees from Java; these libraries are provided

[1] Simple Object Access Protocol

[2] Document Object Model

by Sun Microsystems, and by a variety of open source projects. DOM is considered the standard interface: most other XML APIs include methods for translating their format to and from DOM.

To send a DOM XML tree, one can either construct the tree under program control, or read the XML from a file. The DOM example provided in this article reads the XML from a file following this processing:

1. Create a DOM object from XML stored in a file
2. Send the DOM to the SOAP Server
3. Receive the DOM response
4. Export response data to an Excel spreadsheet

Marshaled Messages

Marshaling is the process of transforming a stream of data into an object. Axis can automatically create classes that correspond to the request and response messages given in the WSDL. In other words, Axis will automatically translate object-based request data into XML, and will translate the response XML back into objects.

Axis includes tools that automatically generate a set of "bean" classes corresponding to the WSDL for the Web Service. The bean classes are simple classes that have "get" and "set" methods corresponding the elements defined in the WSDL. Axis also generates methods that call the individual services defined in the WSDL. These tools make it easy to call services using standard Java techniques.

In the marshaling example, you will first uses Axis tools to automatically generate beans classes and service methods. The provided sample client program uses the generated classes following this processing:

1. Call a service method passing data as a parameter
2. Receive the response as a bean
3. At this point the program is free to process the bean object as needed.

Processing the Response

Whether using DOM or marshaled messages, the SOAP response can be processed under program control. Examples include using key data in the response to read or write data to a database, or writing data to a file for later processing.

The DOM client example writes response data to an Excel Spreadsheet file (XSL) using classes from the open source project POI-HSSF (http://jakarta.apache.org/poi/).

Choosing Between DOM-style and Marshaling-style

Use a DOM-style service when you need to process *XML*, or if you need to provide the data to another process that requires XML. For example, you would choose DOM-style if you need to transform the XML using XSLT.

Use a marshaling-style client when you need to process the data *from within a Java application*. Since Axis provides the data as Java bean objects, you have the power of the Java language to process the data. For example, the JDBC library can be used to write the data to a database.

What Information Are We Trying to Get?

The eRETR server sends data in response to requests coming from filers, county officials, assessors, and DOR staff. In this article we'll use the service that sends RETR data for a range of dates. For example, the property lister may ask to receive a list of RETRs filed during the past week.

Request Data

To get the data for a date range, the request must specify the county, a start date, and an end date. For example, the request may specify county 13 (Dane) for the date range December 1, 2004 through December 31, 2004.

The Web service is a computer application, so needs the data in a specific form. Web services use XML, and the XML format is described in the service's WSDL. For the eRETR service, the request must be formatted like this:

```
1 <ns0: getRecordedERETRByDateRangexmlns:ns0="urn:model.retr.dor">
2   <countyID>13</countyID>
3   <dates>
4     <startDate>2004-12-01</startDate>
5     <endDate>2004-12-31</endDate>
6   </dates>
7 </ns0: getRecordedERETRByDateRange>
```

Figure 1. Request XML — Return RETRs for a Date Range

In figure 1, our request to get data for county 13 is found in line 2 with the "getRecordedERETRByDateRange" tag. The surrounding "countyID" tag identifies the data. The date range is found in lines 3 to 6. The "ns0" in line 1 is a reference to where this XML structure is defined.

This entire request is embedded in a "SOAP Envelope"–this is the standard structure expected by a SOAP service:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <SOAP-ENV:Envelope xmlns:SOAP
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
   <ns0:getRecordedERETRByDocumentIDRange xmlns:ns0="urn:model.retr.dor">
    <countyID>13</countyID>
     <docIDs>
      <startNumber>2654</startNumber>
      <endNumber>2656</endNumber>
     </docIDs>
   </ns0:getRecordedERETRByDocumentIDRange>
   </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Figure 2. A request SOAP Envelope

Response Data

In response to this request the server returns all the returns for that county for the date range. The response is embedded in a SOAP Envelope.

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <getRecordedERETRByDocumentIDRangeResponse xmlns="urn:model.retr.dor">
  <result xmlns="">
    <item>
      <Grantor>...</Grantor>
      <Grantee>...</Grantee>
      <PropertyTransferred >...</PropertyTransferred >
      <FeeComputation>...</FeeComputation>
      <PropertyTransferred >...</PropertyTransferred >
      <RecordingInformation>
       <DocumentNumber>2654</DocumentNumber>
       <CountyCode>13</CountyCode>
      </RecordingInformation>
      ...
    </item>
    <item>
      <Grantor>...</Grantor>
      <Grantee>...</Grantee>
      <PropertyTransferred >...</PropertyTransferred >
      <FeeComputation>...</FeeComputation>
      <PropertyTransferred >...</PropertyTransferred >
      <RecordingInformation>
       <DocumentNumber>2655</DocumentNumber>
       <CountyCode>13</CountyCode>
      </RecordingInformation>
      ...
    </item>
    </result>
    </getRecordedERETRByDateRangeResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Figure 3. A response SOAP Envelope

In this example, the data for two RETRs was sent in response. (An eRETR is lengthy; therefore, this example omits some detail from the response.)

Setting Up the Java Samples

All code and required libraries are in a single ZIP file found at http://www.dor.state.wi.us/eretr/. Download this file and unzip it to a convenient location on your local file system.

`SOAPMessageBodies`

Directory containing XML files containing sample SOAP message bodies.

`bin/`

Directory where the compile batch files write Java binaries for the three Java classes found in `src/`.

`lib/`

Directory of JAR files acquired from the Axis, JDOM, and POI projects.

src/

Java source directory. There are three Java source files:

- sample.client.dom.Client.java
- sample.client.marshaled.Client.java
- sample.xlswriter.NodeWriter.java

`CompileDOMSample.bat`

A batch file that compiles sample.client.dom.Client.java and sample.xlswriter.NodeWriter.java

`CompileMarshalSample.bat`

A batch file that compiles sample.client.marshaled.Client.java

`RunSampleClientDOM.bat`

A batch file that runs the sample DOM client.

`RunSampleClientMarshaling.bat`

A batch file that runs the sample marshaling client.

`RunWSDL2Java.bat`

A batch file that runs the Axis WSDL2Java utility.

Figure 4. Contents of the Java sample ZIP file

Important: All JAR files in lib must be in your Java CLASSPATH. Also note that `sample.client.marshaled.Client` will not compile until you follow the steps in section "Creating an Axis Marshaling Client" below. This will not affect your ability to run the DOM example.

About the Batch Files

The batch files are written assuming your working directory is the location of the file. In other words, the batch files assume you have used DOS commands to navigate to the location of the batch files and are running them from there.

About the Libraries

The sample contains all the libraries you'll need. Three sets of libraries are used: those required by Axis JDOM and POI-HSSF.

The Axis libraries are available at: http://ws.apache.org/axis/. As of November 22, 2004, the current release of Axis is 1.2 RC1. Axis is distributed as a single compressed file. The Axis 1.2 RC1 library files are:

- `axis.jar`
- `axis-ant.jar`
- `commons-discovery.jar`
- `commons-logging.jar`
- `jaxrpc.jar`
- `log4j-1.2.8.jar`
- `saaj.jar`
- `wsdl4j.jar`
- `xmlParserAPIs.jar`

JDOM (http://jdom.org/) is an open source project that provides Java classes for manipulating XML. The JDOM binaries are found at http://jdom.org/downloads/. As of November 30, 2004, the current release of JDOM is 1.0. JDOM is distributed as a single compressed file. The JDOM 1.0 library files are:

- `ant.jar`
- `jaxen-core.jar`
- `jaxen-jdom.jar`
- `jdom.jar`
- `saxpath.jar`
- `xalan.jar`
- `xerces.jar`
- `xml-apis.jar`

The POI binaries are found at http://jakarta.apache.org/poi/. As of November 30, 2004, the current release of POI is 2.5.1. POI is distributed as a single compressed file named `poi-bin-2.5.1-final-20040804.zip`. The POI 2.5.1 library file is:

- `poi-2.5.1-final-20040804.jar`

The Axis DOM Client Sample

The sample Axis DOM client is `sample.client.dom.SendSpecifiedBody`. This is a Java application that reads three parameters:

- The SOAP Server URL
- The full path name of the XML file containing the SOAP message body
- The full path name of the Excel spreadsheet file that will contain response data

If you inspect the files in directory `SOAPMessageBodies/` you'll see that they contain message bodies for some of the services specified in the WSDL. Feel free to copy and edit these to try out other services or to see how the services respond when different data is used.

Compiling the Sample DOM Client

To compile the DOM sample run `CompileDOMSample.bat.` The batch file assumes your working directory is set to the location of the batch file. Running the batch file should result in the creation of `bin/client/sample/dom/Client.class` and `bin/client/xlswriter/NodeWriter.class`.

Running the Sample DOM Client

The batch `RunSampleClientDOM.bat` file runs the main method in `sample.client.dom.SendSpecifiedBody` passing these values:

- The SOAP server URL is set to the name of the server,
- The XML file is set to `/SOAPMessageBodies/GetRetrList.xml`
- The output XLS files is set to `C:\temp.xls`

Running the batch file should result in the creation of the spreadsheet.

The sample uses class `sample.xlswrite.NodeWriter` to write DOM data to an Excel spreadsheet. `NodeWriter` uses POI-HSSF to create the spreadsheet. The topic of using POI-HSSF is beyond the scope of this article. However, feel free to browse the source of `sample.xlswrite.NodeWriter`. Information on POI can be found in the "Sources of Information" section at the end of this article.

The source file contains two methods: `main`, and `callService`. Method `callService` does all the work.

```
public static void main(String[] args) throws Exception {
  callService(args[0], args[1], args[2]);
}

public static void callService(String serviceURL,
  String soapBodyFileName, String fileName) throws Exception {

  // 1. Create an array of SOAPBodyElements, one array element for
  // each element in the input document.
  // 2. Call the service passing the array. The response is a vector
  // of SOAPBodyElements.
  // 3. Print the leaf values of each returned element to an Excel
  // spreadsheet.

  // 1.
  InputStream inputStream = new FileInputStream(soapBodyFileName);
  Document inputDocument = XMLUtils.newDocument(inputStream);
  NodeList nodeList = inputDocument.getChildNodes();
  SOAPBodyElement[] input = new SOAPBodyElement[nodeList.getLength()];
  for (int i = 0; i < nodeList.getLength(); i++) {
    Element anElement = (Element) nodeList.item(i);
    input[i] = new SOAPBodyElement(anElement);
  }
  // 2.
  Service service = new Service();
  Call call = (Call) service.createCall();
  call.setTargetEndpointAddress(serviceURL);
  Vector response = (Vector) call.invoke(input);
  int f = 0; // File name index
  for (Iterator it = response.iterator(); it.hasNext();) {
    MessageElement anElement = (MessageElement) it.next();
    // 3.
    Element domElement = anElement.getAsDOM();
    // Sorry about the ugly expression to add _# to each file name :(
    String fName = fileName + (f++==0?"":"_"+f);
    // Use xpath to get the "result/item" elements. This matches what
    // our example returns; change the xpath if you change examples.
    NodeList nl = XPathAPI.selectNodeList(domElement, "//result/item");
    NodeWriter.writeNodeLeaves(fName+".xls", nl);
    FileOutputStream fileOut = new FileOutputStream(fName+".xml");
    XMLUtils.PrettyElementToStream(domElement, fileOut);
  }
}
```

Figure 5. The two methods in sample.client.dom.SendSpecifiedBody

What Does This Code Do?

Section 1 reads the SOAP message body from the specified XML file and uses it to create a DOM object. The code then initializes an array of `org.apache.axis.message.SOAPBodyElement` objects. Each array element corresponds to an element within the sample message. Axis will place these elements into the message it sends to the server.

Section 2 starts by getting an `org.apache.axis.clientService` object. This object encapsulates a Web Service. The code then gets a `Call` object from the service and has the `Call` reference the server name passed into the routine. The `Call` object's invoke method has Axis place the elements in a SOAP message, pass them to the server, and receive a response. (RETR response messages always contain a single response element, which in turn may contain child elements.)

In **section 3**, the response is translated back into a DOM object, and the `NodeWriter` class called to write the result to an Excel spreadsheet. The `NodeWriter` source is included with the sample. It uses POI-HSSF classes to create the Excel spreadsheet file. Note that section 3 passes specified elements to `NoteWriter`. These are specified in an XPath expression that matches what's being returned in this example. If you were to use a different SOAP request, you would change the XPath expression to select the elements you want to export to Excel.

Creating an Axis Marshaling Client

To use the Axis marshaling features you need to run an Axis utility named WSDL2Java. This utility generates bean classes based on the Web Service WSDL, and generates a method for each service.

Running WSDL2Java

The eRETR samples include a batch file that runs WSDL2Java.

```
@ECHO OFF
SET CP=.
SET CP=%CP%;lib/axis.jar
SET CP=%CP%;lib/axis-ant.jar
SET CP=%CP%;lib/commons-discovery.jar
SET CP=%CP%;lib/commons-logging.jar
SET CP=%CP%;lib/jaxrpc.jar
SET CP=%CP%;lib/log4j-1.2.8.jar
SET CP=%CP%;lib/saaj.jar
SET CP=%CP%;lib/wsdl4j.jar
SET CP=%CP%;lib/xmlParserAPIs.jar

SET WSDL=http://rvpc3870:9080/RetrWebServices/services/RetrServices2?wsdl
@ECHO ON

java -cp %CP% org.apache.axis.wsdl.WSDL2Java --verbose --output src --
noWrapped %WSDL%
```

Figure 6. The RunWSDL2Java.bat batch file

Note that the batch file passes the name of the Web Service WSDL to WSDL2Java. The WSDL contains information needed by WSDL2Java, such as the service URL, available services and the structure of the messages being used by the services.

Running the batch file results in WSDL2Java creating a package named for the service URL, a set of bean classes corresponding to the message elements, and a "service" class with a set of methods that allow the programmer to send and receive messages to the various services. Specifically, running `RunWSDL2Java.bat` will cause WSDL2Java to generate these Java source files:

- Many "bean" classes corresponding to the message element types
- `dor.retr.model.ERETRPort`, which is an interface whose methods correspond to the operations provided by the Web Service service (using the "operation" elements from the WSDL)
- `dor.retr.model.ERETRService`, which is an interface that returns an ERETRPort implementation.
- `dor.retr.model.ERETRServiceLocator`, which is the implementation of `ERETRService`
- `dor.retr.model.ERETRPortSOAPBindingStub`, which is the implementation of `ERETRPort`

Compiling the Sample Marshaling Client

To compile the marshaling sample run `CompileMarshalingSample.bat`. The batch file assumes your working directory is set to the location of the batch file. Running the batch file should result in the creation of `bin/client/sample/marshaling/Client.class`.

Running the Sample Marshaling Client

The batch `RunSampleClientMarshaling.bat` file runs the main method of `client.sample.marshaling.Client`. This class calls.

```
public static void main(String[] args)
  throws ServiceException, RemoteException {

  // Get the object that has the list of services.
  ERETRPort retrPort = new ERETRServiceLocator().geteRETRPort();

  // Call the service that retrieves a list of eRETRs. The parameters are
  // county ID, and DateRange--a start and end date. First set up the
  // parameters.
  String countyCode = "13";
  DateRange dateRange = new DateRange();
  Calendar c = new GregorianCalendar(2004, GregorianCalendar.DECEMBER, 1);
  dateRange.setStartDate(c.getTime());
  c = new GregorianCalendar(2004, GregorianCalendar.DECEMBER, 31);
  dateRange.setEndDate(c.getTime());

  // Now that the parameters are set up, call the service, get a
  // response, and print it.
  RecordedRETR[] retrArray =
    retrPort.getRecordsByDates(countyCode, dateRange).getItem();
  for (int i = 0; i < retrArray.length; i++) {
    RecordedRETR r = retrArray[i];
    System.out.println(r.getRecordingInformation().getDocumentNumber());
  }
}
```

Figure 7. The sample.client.marshaling.Client main method

What does this code do?

The code calls uses the `ERETRServiceLocator()` method `geteRETRPort()` to get an object that implements `ERETRPort`; this object encapsulates the details of connecting to the server and translating parameters sent as objects into the XML that is actually sent to the service.

The service expects complex data—a county code and a structured element type that stores start and end dates—the service method parameters match those types. Specifically, `getRecordsByDates` expects the county code to be passed as a `String`, and the start and end dates to be passed as a `DateRange`. The `DateRange` type was automatically created by Axis when you ran WSDL2Java. Therefore, to call the service the code stores the county code in a String and creates a DateRange object, and passes the objects as parameters to the getRecordsByDates method. This method returns an element that contains an array of eRETRs. Again—the return type of this method was determined by the information found in the WSDL.

This code is differs from the DOM example because the generated classes encapsulate the format of the request and response as bean objects, and hides the details of making the call to the server.

Sources of Information

Spreadsheet Applications

Excel is not required to use XLS files. There are several spreadsheet products that also read XLS files:

- OpenOffice (an open source project sponsored by Sun Microsystems) includes a spreadsheet application. See http://www.openoffice.org/ for details.
- Lotus 123 (IBM Corporation)
- Quatro (Corel Corporation)

Note that Microsoft Works (Microsoft Corporation) does *not* read XLS files.

Apache Axis

The Axis project home page is http://ws.apache.org/axis/.

The primary programming reference is the *Axis User's Guide*, found at http://ws.apache.org/axis/java/user-guide.html.

Ongoing discussions of Axis are found at the Axis Wiki, which is linked off of the general Apache Wiki at http://wiki.apache.org/general.

A list of books and articles relating to Axis is found at http://ws.apache.org/axis/java/reading.html.

JDOM

The JDOM project home page is http://jdom.org/.

POI-HSSF

The POI project home page is http://jakarta.apache.org/poi/index.html.

The HSSF project home page is http://jakarta.apache.org/poi/hssf/index.html.

There is an article on POI-HSSF at IBM's Developer Works: http://www-106.ibm.com/developerworks/db2/library/techarticle/dm-0402bhogal/.

Glossary

Apache Software Foundation

According to the Apache Software Foundation home page:

The Apache Software Foundation provides support for the Apache community of open-source software projects. The Apache projects are characterized by a collaborative, consensus based development process, an open and pragmatic software license, and a desire to create high quality software that leads the way in its field. We consider ourselves not simply a group of projects sharing a server, but rather a community of developers and users.

DOM

Document Object Model. An open source API for processing XML trees as objects.

Jakarta Project

According the Jakarta Project home page:

The Jakarta Project creates and maintains open source solutions on the Java platform for distribution to the public at no charge.

JDOM

Java Document Object Model. JDOM is an open source set of Java classes that process XML and DOM.

Marshaling

An informal term referring to the translation of an object to and from some other format. Typically, the object is transformed to and from an XML representation of the object. In general, *marshal* is synonymous to *serialize*.

Open source

Open source refers to application software whose source is made freely available. Open source software is usually free.

POI-HSSF

Poor Obfuscation Implementation, Horrible Spread Sheet Format. POI is an Apache-Jakarta project whose family of Java classes allow the programmer to process some Microsoft Office file formats.

SOAP

Simple Object Access Protocol. SOAP is a standard format for sending and receiving Web Services messages. In common usage, SOAP is synonymous with Web Services.

SOAP Message

A SOAP message has the format:

```
<Envelope>
  <Header>
  </Header>
  <Body>
  </Body>
</Envelope>
```

Data is sent in the message body. (Meta-data is sent in the header.)

Web Service

A server that can receives XML-based requests and sends XML-based responses.

A Web Service is analogous to a Web Application. Web Applications receive requests from a Web browser and send HTML responses. The user is a person using a browser such as Internet Explorer. Similarly, a Web Service receives requests and sends responses. However, in this case, the "user" is a computer application.

WSDL

Web Services Description Language. A WSDL fully describes a Web Service. For example, it names the service URL, the services (operations) provided, and the format of the request and response XML.

XML

Extensible Markup Language. An XML file is a file containing tags and associated data.

Historically, XML came about is a rigorous version of HTML. HTML is inconsistent in its use of tags. For example, not all HTML tags must be terminated. This led to the need for a well-formed markup format: XML.

XPath

XML Path Language (XPath) is a language for addressing parts of an XML document.